Australian Government
Department of Defence
Defence Science and
Technology Organisation

Secure Ad Hoc Networking on an Android Platform

Angus Morton, David Adie, Paul Montague

Cyber and Electronic Warfare Division

Defence Science and Technology Organisation

DSTO–TN–1390

ABSTRACT

This document describes and analyses a suite of applications for the Android platform that enables easy and secure ad hoc networking, with a focus on extensibility and reusability. The applications were developed by the first author during a Swinburne University Industry-Based Learning placement at the Defence Science and Technology Organisation, under the latter's Industry Experience Placement Program.

Published by

DSTO Defence Science and Technology Organisation
PO Box 1500
Edinburgh, South Australia 5111, Australia

Telephone: 1300 333 362
Facsimile: (08) 7389 6567

AR No. 016-201
May, 2014

Secure Ad Hoc Networking on an Android Platform

Executive Summary

Android is a quickly maturing open source platform that is being widely adopted by individuals and by organisations (including Defence). This document describes a prototype implementation of a secure ad hoc networking system for Commercial Off The Shelf (COTS) Android platforms with a focus on extensibility and security. This implementation utilises Near Field Communication (NFC) and Bluetooth to create secure, mutually authenticated ad hoc networks between devices for the sharing of information. The development work was conducted during a Swinburne University Industry-Based Learning (IBL) placement at the Defence Science and Technology Organisation (DSTO), under the latter's Industry Experience Placement (IEP) Program.

One of the core issues surrounding Android involves the security of the operating system. Security-Enhanced Android (SE Android) aims to mitigate some of these issues by applying Mandatory Access Control (MAC) to Android. SE Android's MAC is currently applied to Android 4.4, however it is in permissive mode for most applications, and enforcing for root level applications only. In addition to the implicit protections provided by SE Android, we utilised SE Android's MAC to assist in securing our applications through the use of a global security policy. The policy restricts the permissions granted to our applications and other applications on the devices.

In order to enable mutual authentication each device is provisioned with a certificate that is signed by the Device Issuing Authority (DIA). The DIA provides the root of trust that ensures the legitimacy of the devices. The certificate provided by the DIA includes information about the owner of the device, which may be one or more of:

- name
- address
- date of birth
- ID card number
- portrait of owner.

Once the device is provisioned, connections can be initiated by tapping two devices together, this uses NFC to bootstrap a Bluetooth connection. Once the two devices are connected via Bluetooth, the station-to-station protocol is run which mutually authenticates the devices to one another using the DIA signed certificates. These connections are maintained by a session manager on the Android platform that allows new applications to be written that make use of this secure connectivity capability.

THIS PAGE IS INTENTIONALLY BLANK

Contents

Appendices

Abbreviations

ADB	Android Debug Bridge
AES	Advanced Encryption Standard
AOSP	Android Open Source Project
API	Application Programming Interface
ASLR	Address Space Layout Randomisation
BLE	Bluetooth Low Energy
CA	Certificate Authority
COTS	Commercial Off The Shelf
CPU	Central Processing Unit
CSR	Certificate Signing Request
DARPA	Defense Advanced Research Projects Agency
DIA	Device Issuing Authority
DoS	Denial of Service
DSTO	Defence Science and Technology Organisation
ECDSA	Elliptic Curve Digital Signature Algorithm
GPS	Global Positioning System
IBL	Industry-Based Learning
ID	Identifier
IEP	Industry Experience Placement
IPC	Inter-Process Communication
IR	Infrared
MAC	Mandatory Access Control
MITM	Man-In-The-Middle
MMAC	Middleware Mandatory Access Control
NFC	Near Field Communication
NSA	National Security Agency
NX	No eXecute

OS	Operating System
P2P	Peer-to-Peer
PIN	Personal Identification Number
RAM	Random-Access Memory
RF	Radio Frequency
SE	Secure Element
SE Android	Security-Enhanced Android
SMS	Short Message Service
SSP	Simple Secure Pairing
TPM	Trusted Platform Module
USB	Universal Serial Bus
UUID	Universally Unique IDentifier

1 Introduction

There has been significant development in the area of secure mobile computing recently, including the development of commercial off the shelf (COTS) Android secure platforms such as Samsung's KNOX[1] and ESD's Cryptophone 500[2]. Android can lever many useful hardware features which the majority of devices support, including:

- location sensing, especially via Global Positioning System (GPS)
- hardware-backed credential storage
- wired connectivity *i.e.* Universal Serial Bus (USB) connectivity
- wireless connectivity *i.e.* Bluetooth, Wi-Fi, Near Field Communication (NFC).

In addition to supporting these features, Android is widely adopted and is being actively used for development by many governments and defence organisations around the world (*e.g.* the National Security Agency's (NSA) Security-Enhanced Android (SE Android) [Smalley 2011], the Defense Advanced Research Projects Agency's (DARPA) Mobile Armor [Franceschi-Bicchierai 2012]), making it an attractive platform to develop with.

This report discusses the current state of security on Android, and a secure implementation for creating ad hoc networks among multiple devices with a view for extensibility. The implementation is intended to provide a base which other applications can build off and utilise easily and securely. The document also discusses both the security benefits and drawbacks of using the proposed system. Furthermore there has been a focus on keeping the devices COTS where possible, with the only modification made to the platform being the inclusion of SE Android's Install-time Mandatory Access Control (MAC) feature.

Some knowledge of Android is required to understand the system described in this document. *Getting Started* [2013] provides an overview of the Android platform, as well as in-depth tutorials on the relevant features of Android.

2 Android Security

This section will focus on the Platform Security Architecture, as described in *Android Security Overview* [2013]. A more comprehensive discussion of Android security is provided by Gleeson & Lucas [2013].

The Android kernel has many differences to that of Linux, which are driven not only by the need to support the additional hardware present in mobile devices, but also by the removal of some Linux kernel features as part of a process of simplification – thus reducing the attack surface [Fledel et al. 2012]. The overall architecture of the Android software stack is given in Figure 1 for ease of reference.

The main security features of the Android operating system (OS) include:

[1]http://www.samsung.com/global/business/mobile/solution/security/samsung-knox
[2]http://esdcryptophone.com/pdf/Brochure-CP500.pdf

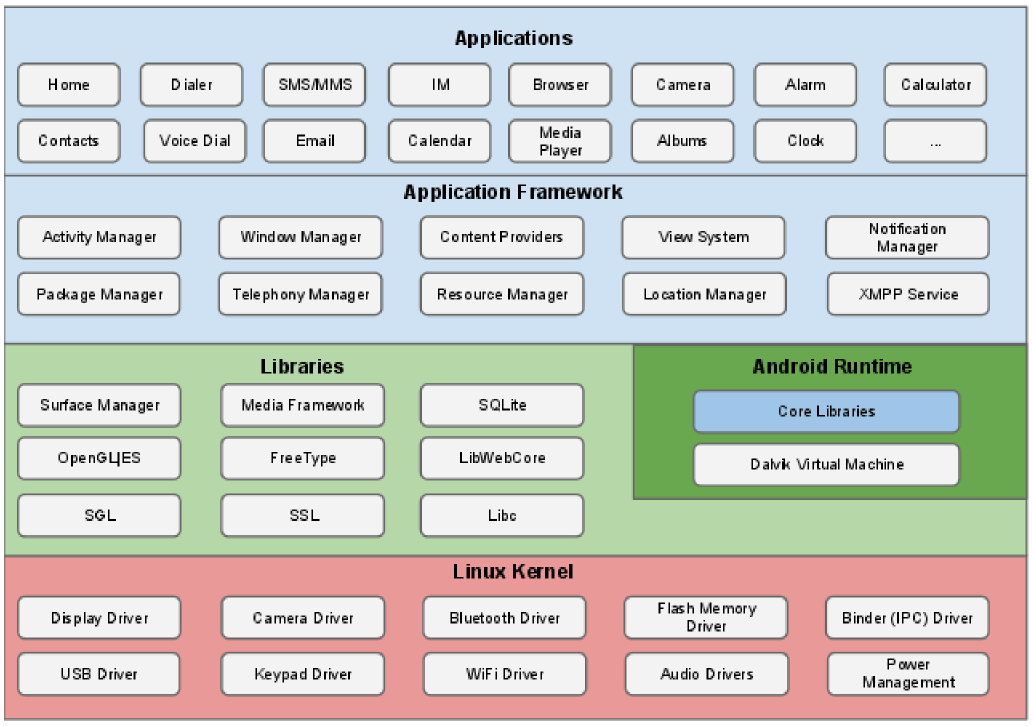

Figure 1: *Android Software Stack [*Android Security Overview *2013].*

- Application Sandbox – All applications above the Linux Kernel layer (per Figure 1), even native code and operating system applications, are sandboxed by the kernel layer sandbox support. This means that applications cannot easily access/modify the code/data of other applications [*Android Security Overview* 2013]. Sandboxing on Android has the following properties:
  - Unique user ID – each application is assigned a unique user ID on installation which is used to enforce the sandboxing [Khan et al. 2010]. However two applications can optionally share an ID if they are signed by the same developer.
  - Unique processes – each application typically runs in its own process. However if two applications are signed by the same developer they can optionally be run in the same process [Khan et al. 2010].
  - Install-time Permissions – In order to access resources, Android applications must request the required permissions for that resource (as specified in their AndroidManifest.xml) at install time. The user is responsible for accepting or rejecting permissions [*The AndroidManifest.xml File* 2013, Khan et al. 2010].
  - Data Security – By default files created by an application are owned by that application's unique ID and are, therefore, only readable or writable by that application [Liebergeld & Lange 2013]
- Code Integrity:
  - Signed Applications – Applications must be signed by the developer's private key. Note that the developer's certificate may be self-signed [*Android Security Overview* 2013].

  - Application Verification – The Package Manager verifies that the application has been signed and checks its signature against the included certificate [*Android Security Overview* 2013].
  - Read-only System – The system partition that contains the application framework, application runtime, and shared libraries is mounted as read-only by default [Liebergeld & Lange 2013].

- Memory management security: the kernel includes memory management security features such as hardware-based No eXecute (NX) to prevent code on the stack/heap being executed, as well as Address Space Layout Randomisation (ASLR) and format string vulnerability protections.

- Filesystem encryption: from Android 3.0 onwards, full filesystem encryption (using AES128 and SHA256) is supported [*Android Security Overview* 2013]. User data is encrypted using a key derived from a user password.

2.1 SE Android

SE Android [Smalley 2012] is based on SELinux, which is a version of Linux enhanced with Mandatory Access Control (MAC) in order to enforce a system wide security policy. User programs and system daemons are confined by SELinux to the least amount of privilege required to fulfil their role, thus limiting the amount of damage which they may inflict if compromised [Loscocco & Smalley 2001]. SE Android applies the same concept to Android, in order to:

- confine privileged daemons
- sandbox and isolate applications more strongly than the standard Android mechanisms (including preventing privilege escalation)
- provide a fine-grained and centralised security policy.

The NSA's motivation [Smalley 2013, Smalley 2012] for the development of SE Android arose from the increasing desire to use mobile devices within government, recognition of the widespread adoption of Android in the market and an overall need for improved security in mobile operating systems. The goal of the SE Android project was "to identify and address critical gaps in the security of Android" [Smalley 2013], initially by applying SELinux to Android, though the scope of the project is not limited to this.

The NSA note that SE Android is not a government-specific version of Android, nor is it evaluated or approved in any way [Smalley 2012]. Rather, it is a set of security enhancements that focusses on security gaps of wide applicability and "targeting mainline Android adoption" [Smalley 2013].

2.1.1 Adoption of SE Android

The decision to utilise SE Android was supported by the gradual introduction of SE Android into the Android Open Source Project (AOSP) mainline. AOSP now utilises the SE Android kernel modifications for MAC. As of Android 4.4, SE Android runs in enforcing mode for the root domain and root level applications, though permissive mode is still applied by default for the application domain within which user applications reside [*Validating Security-Enhanced Linux in Android* 2013]. In permissive mode, which 4.3 used globally, policy violations are logged but not denied.

Other SE Android functionality still remains to be activated in stock versions of Android – notably Middleware Mandatory Access Control (MMAC) – without which the secure management of inter-app communications by the Android OS is limited [Smalley & Craig 2013]. Additionally, the policy used by default in Android 4.3 and 4.4 is liberal in order to limit the number of applications that are blocked. However, it is anticipated that a stronger policy will be introduced as well as SE Android MMAC in future versions of Android.

2.1.2 Middleware MAC

SE Android provides, in addition to kernel level protection, middleware MAC (MMAC) support for a small set of features (at the time of writing). Full MMAC is required for the Android platform because not all attacks can be contained at the kernel level (*e.g.* Permission re-delegation [Felt et al. 2011]).

The SE Android project included a number of experimental solutions in the past for handling MMAC. However the only three continuing and ongoing solutions are Install-time MAC, Enterprise Operations, and the more recent IntentFirewall[3] (introduced in Android 4.3). It is expected that the required functionality from the experimental solutions will be introduced in the future using a combination of Enterprise Operations and the IntentFirewall, and that these features will subsequently be introduced into mainline AOSP.

2.2 Residual Issues

Whilst SE Android mitigates many of the residual security issues of stock Android, some vulnerabilities still remain, for example:

- kernel vulnerabilities [Smalley 2012]
- poor security policy – a strong security policy and security-focused application architecture are critical
- recovery or other boot-chain related methods [Fora 2013]
- malicious hardware [Beaumont, Hopkins & Newby 2011].

[3]http://selinuxproject.org/page/SEforAndroid

3 Device Provisioning

In order to be able to identify an individual and their device in the field, the device must be provisioned with some identifying details to distinguish it and its owner. In this example we use a basic X.509 certificate with the owner's name as the identifying feature, however additional information could be included in the certificate, for example:

- portrait of device owner[4]
- date of birth
- ID number
- home address.

The provisioning process is undertaken in a secure environment with a trusted host. Once a device is provisioned it is able to identify itself to another device, and each device is able to prove that it is who it claims to be. The implementation of the provisioning process is discussed in detail in Section 3.1.

The protocol for the device provisioning is as follows:

1. Host $\rightarrow$ Device: Relevant certificate details (e.g. full name, date of birth, organisation ID number, photo).
2. Device: Create private-public key pair (stored on hardware key storage if available).
3. Device: Create certificate signing request (CSR) using generated key pair.
4. Device $\rightarrow$ Host: Certificate signing request containing the relevant details.
5. Host: Creates Certificate Authority (CA) signed certificate from the CSR.
6. Host $\rightarrow$ Device: CA signed device certificate and CA certificate for verification.
7. Device: Verify that the device certificate was signed by the CA and includes all supplied details correctly.

3.1 Implementation

3.1.1 Host

The host in the implemented prototype runs a Java application, whose role in the provisioning process is to provide a trusted certificate issuing authority as well as any extra information like photos of the owner, biometric data etc. In the prototype implementation the host will be referred to as the Device Issuing Authority (DIA).

For this prototype implementation the DIA accepts a connection over the Android Debug Bridge (ADB) using the port forwarding mechanism [*Android Debug Bridge* 2013].

[4]RFC 6170, http://www.rfc-base.org/txt/rfc-6170.txt

Note that in a production environment the Android Accessory Communication Protocol[5] is the preferred method for communication from host to device.

The private-public CA key pair are 2048 bit RSA keys. The keys were generated using the Java keytool command to create a Java keystore file (.jks), the command used can be found in Appendix A. As of Android 4.4 support for elliptic curve keys has been extended to the Android key store, which means that the keys generated can optionally be elliptic curve digital signature algorithm (ECDSA) keys.

When the Host receives the CSR it programmatically generates and signs a certificate. Relevant parts of the Java code used in the prototype are shown in Appendix B.

3.1.2 Device

The device-side of this implementation is an Android application made up of three components (as highlighted in Figure 2). Each of these components performs a specific role as described below:

- Application – Enforces the application logic and handles the connection to the Host
- Signing Service – Stores the device private key and device certificate, and signs objects with the device private key. Note that the private key never leaves the application boundary of the Signing Service.
- Verifying Service – Stores the CA certificate, device certificate, and any other relevant certificates that are needed. Verifies objects and certificates were signed by the proper entity.

The application is designed such that services can be re-used in the future. In addition, it allows delineation and separate accreditation of the code handling sensitive data (Signing Service).

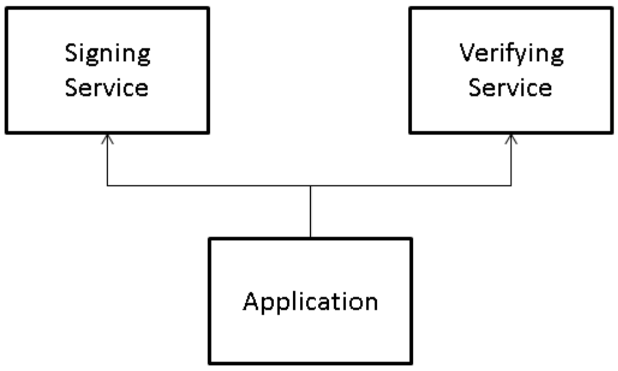

Figure 2: *Device-side application architecture highlighting sandbox separation.*

[5]http://developer.android.com/guide/topics/connectivity/usb/index.html

The Signing Service makes use of the Android Keystore, which was introduced in Android 4.3 to handle application specific keys. The Android Keystore is a security provider that will automatically generate hardware-backed keys if a hardware-backed keystore is available (Secure Element, Trusted Platform Module (TPM), or TrustZone) [Elenkov 2013]. Hardware-backed keys are bound to the hardware and are unable to be read even at the kernel layer. Note that on TrustZone-based devices, keys are encrypted with a hardware-bound master key then stored on the file-system. This allows any number of keys to be protected by the hardware.

Given that the private key is secured by hardware and resistant to exfiltration it is important to protect the Signing Service in order to prevent it from performing operations for an adversary inadvertently. To this end the Signing Service is protected by two Android permissions, one for adding new key pairs and the other for signing objects. It is important to note that we do not declare the permissions as signature level to avoid potentially dangerous permission combinations by other applications signed by the same signer. (This is because when a permission is declared with the signature level of protection it is automatically granted to all applications signed by the same signer [*Permission Element* 2013].) Instead we rely on SE Android and a custom security policy that prevents other applications from being granted our permissions. See Appendix C for an example SE Android install-time MAC policy that protects our permissions.

3.2 Security Discussion

Since the provisioning process is the foundation upon which the rest of the trusted interactions are built, it is important to ensure it is secure. As such there are a number of security considerations to take into account during device provisioning:

- In the case of where there are no secure key storage/generation facilities Android falls back on software storage, and the keys are stored on the file system encrypted with a key derived from the device unlock code. The Android Application Programming Interface (API) supports a method called 'isHardwareBacked()' which returns true if key storage and generation is hardware secured.
- Without install-time MAC it is difficult to ensure a malicious application cannot be granted permission to use the Signing Service (Note that signature level permissions are not used).
- Most Android devices on the market support TrustZone – Vulnerabilities in Motorola's TrustZone kernel have been found in the past[Rosenberg 2013], where the boot loader was unlocked to enable the use of system images not signed by the developer.
- Certificate Authority must be secure – If the CA is compromised the root of trust for all of the devices provisioned by the CA is compromised. In this system the CA resides on an air-gapped computer.

In order to mitigate some of these issues related to the provisioning of devices we built AOSP with the SE Android MMAC changes. This decision was driven by the fact that Android has been progressively introducing SE Android into mainline AOSP, so we anticipate that Google will include install-time MAC in the future. This meant we were able to write our own policy that grants the signing service permission to a few select applications only. For an example of the install-time MAC policy used for the signing service see Appendix C.

Judging by the vulnerabilities found in Motorola's TrustZone kernel previously, it may be necessary to move to a more secure solution for storing the sensitive information (including private keys). The ideal solution would be to store the keys on a dedicated and tamper-resistant hardware module like a Secure Element (SE), though this is typically not possible without vendor assistance because only the device vendor has access to the SE.

4 Connectivity

One of the key benefits of the Android platform is the number of features it supports off the shelf, including a wide selection of connectivity capabilities. These generally include Bluetooth, Near Field Communication (NFC), WiFi, Peer-to-Peer WiFi, as well as Infrared (IR) Communication in the future[6]. This section discusses a selection of connectivity capabilities utilised by the prototype, including potential security benefits, issues, and solutions associated with their use.

4.1 NFC

NFC is a very short-range (up to 10 cm) radio communication technology [Agrawal & Bhuraria 2012], that supports communication in three different ways:

- Card Emulation – The NFC chip acts like a contactless smart card.
- Reader Emulation – The NFC chip acts like a card reader to read smart cards.
- Peer-to-Peer – This mode enables two-way communication between two NFC enabled devices.

Each of these operating modes is achieved in a different way, with the initiator and target performing different roles. As shown in Figure 3 each device can be either active or passive. If a device is active it is creating a radio frequency (RF) field whereas if it is passive it is not creating an RF field, and instead uses inductive coupling to communicate [Van Damme, Wouters & Preneel 2009].

When the NFC feature was introduced in Android 2.3[7] it was relatively limited in that it only supported two modes: peer-to-peer, and reader emulation. Furthermore, the API is relatively limited in what it allows programmers to achieve using peer-to-peer mode, in that they can only perform an NFC 'NDEF beam' to transmit a single block of data one

[6]http://developer.android.com/guide/topics/connectivity/index.html
[7]http://developer.android.com/about/versions/android-2.3-highlights.html

	Initiator	Target
Card Emulation	Passive	Active
Reader Emulation	Active	Passive
Peer-to-Peer	Active	Active

Figure 3: *NFC operating modes and device RF behaviour.*

way. This has limited the use of NFC for security purposes to using it as a method for bootstrapping another communication channel, as shown in Section 4.3.1.

Although the use of NFC is limited by the current API, it is still an attractive method of communication for a number of reasons:

- Short-range – Simplifies identifying which device you are attempting to communicate with. As opposed to navigating menus to select the intended device out of a list of possible devices.
- Man-in-the-Middle Resistant – Haselsteiner & Breitfuß [2006] determined that stealthy Man-in-the-Middle attacks are infeasible over NFC. Note that relay attacks are still viable, whereby a malicious pass-through device simply passes the NFC messages on to the real device and eavesdrops on the communication.
- User friendly – All NFC requires is an intuitive tap against another NFC enabled device or tag.
- Energy Efficient – The passive side of communications has low to no energy usage.

Some issues with NFC remain, however, for example:

- Quite slow – Expected speeds between 106 kbps, and 424 kbps, making NFC unsuitable for sending large amounts of data[Remedios et al. 2006].
- Eavesdropping – NFC should not be considered a confidential communication channel because it is not encrypted by default. Furthermore Haselsteiner & Breitfuß [2006] suggest the average distance for successful eavesdropping to be between 1m and 10 m, depending on the scenario. Furthermore Kortvedt & Mjølsnes [2009] discovered they could eavesdrop at a distance of 20-30cm using an improvised passive listener without performing any amplification or filtering.
- Denial of Service attacks – Like other wireless communication methods NFC is open to interference.
- Data Modification – Haselsteiner & Breitfuß [2006] determined that this attack is feasible, depending on the strength of the amplitude modulation. In the best-case scenario it is possible to modify certain bits but impossible for others. The worst-case allows modification of all bits. Whether this could be used to meaningfully modify data in real-time is unknown at this stage.

- Data Insertion – This is where data is inserted into the data exchange between two devices by an attacker in place of a response message from the answering device. This is only feasible if the attacker finishes transmitting the inserted packet before the answering device begins transmitting the legitimate reply [Haselsteiner & Breitfuß 2006].

In order to protect against eavesdropping over NFC some form of key agreement protocol would need to be implemented. A standard unauthenticated key agreement protocol, combined with a cipher, would protect against eavesdropping, and could be implemented in the application space. Alternatively, Haselsteiner & Breitfuß [2006] propose an efficient NFC specific key agreement mechanism to eliminate eavesdropping and make meaningful data modification more difficult. Protecting against eavesdropping is important and is discussed further in Section 4.4.1.

4.2 Bluetooth

Bluetooth is a wireless technology standard used to transmit data over short distances by creating ad hoc networks called piconets. A piconet is a set of two or more Bluetooth devices operating on the same channel using the same frequency hopping sequence. Bluetooth uses frequency hopping to reduce interference and transmission errors, as well as provide limited transmission security, since Bluetooth operates on a range of frequencies [Padgette, Scarfone & Chen 2012].

As of Android 4.3 both classic Bluetooth as well as Bluetooth Low Energy are supported. This section, however, only refers to classic Bluetooth as there are some key differences between the classic and low energy variants[8]. For example Bluetooth Low Energy has a much lower data throughput than classic. Furthermore, Bluetooth Low Energy does not provide passive eavesdropping protection, whereas classic does.

Some of the benefits of using Bluetooth for ad hoc networks over other wireless connectivity standards (like Wi-Fi) are:

- Low energy use – Compared to ad hoc Wi-Fi, especially when the Bluetooth device is non-discoverable (the default for Android) [Friedman, Kogan & Krivolapov 2013].
- Short-range – 5 to 9 metre radius of connectivity [Kizza 2013, Padgette, Scarfone & Chen 2012]
- Fast – Theoretical throughput of 2.1 Mbps [Padgette, Scarfone & Chen 2012]
- Power Control – Devices in a piconet can adjust their radio power levels by incrementally increasing or decreasing the transmission power to conserve power and reduce the signal range. [Padgette, Scarfone & Chen 2012]

The combination of radio power control and frequency hopping provide some implicit protection against eavesdropping and malicious access. Although Spill & Bittau [2007]

[8]developer.bluetooth.org/TechnologyOverview/Pages/LE-Security.aspx

describe a method by which they were able to calculate the hopping sequence and, therefore, eavesdrop on the communications (given hardware with the appropriate frequency switching abilities).

Since Bluetooth is a wireless technology it suffers from some of the same weaknesses such as denial of service attacks, eavesdropping, man-in-the-middle attacks, message modification, and so on. One example of a weakness is a theoretical attack described by Lu, Meier & Vaudenay [2005] where they were able to recover the encryption key in 2^{38} computations. This attack could be mitigated by applying an application-layer encryption scheme over the top (such as Diffie-Hellman with an Advanced Encryption Standard (AES) cipher[Menezes, van Oorschot & Vanstone 1996]).

Android makes use of Bluetooth security mode 4, which uses Simple Secure Pairing (SSP) whereby Elliptic-Curve Diffie-Hellman key exchange is performed after a link has been established. This is at odds with Padgette, Scarfone & Chen [2012] who recommend the use of security mode 3 over the other modes because it requires the establishment of authentication and encryption before the physical link is completely established. Unfortunately on Bluetooth 2.1+ devices, security mode 4 is the default mode.

4.3 Implementation

In order to create a mutually authenticated, secure channel with another device, we use a two step process involving NFC and Bluetooth:

1. The two devices are brought into contact and NFC is used to establish a pairing by passing of a Bluetooth MAC address and channel ID. This process of NFC to Bluetooth handover is described further in Section 4.3.1.

2. The devices execute the Station-to-Station protocol via Bluetooth, bootstrapping the secure connection from the NFC contact. This protocol is described further in Section 4.3.2.

This two step process confers a number of benefits over a purely Bluetooth connection process, for example rather than a complex Bluetooth configuration process the user can simply touch the devices together to initiate a connection. Furthermore, it bypasses the Bluetooth discovery protocol, which means that the devices will not respond to discovery queries from external devices.

4.3.1 NFC to Bluetooth Handover

The first part of the process is called handover. This is whereby the connection details get handed to another device, which the other device then uses to initiate a connection. On Android 4.3 the flexibility of this part of the protocol is limited by the API with respect to how the application layer can interact over NFC (see Section 4.1). The NFC part of the protocol sends the following information to the receiver, which the receiver then uses to initiate a Bluetooth session with the sender:

- Device Bluetooth MAC Address – Used by the receiving device to initiate a connection directly, bypassing Bluetooth's discovery process.
- Service Record Universally Unique ID (UUID) – A random 128 bit UUID generated as per RFC4122[9] that describes the service to connect to.

Once the connection information is sent via NFC, the sender begins listening for a connection from the UUID. Once a connection from the UUID is created, it stops listening for connections on that UUID.

4.3.2 Connection Protocol

Once a wireless connection between two devices is established the secure session establishment protocol begins. It is assumed that each device is provisioned with the following data (see Section 3 for provisioning process):

- device certificate, issued by the DIA
- device private key
- DIA certificate.

The underlying protocol for secure session establishment between the devices (A and B), once each has been unlocked by its user via a personal identification number (PIN) (or ideally biometric in future) is (heuristically) as follows:

1. Establish shared secret using Elliptic Curve (EC) Station-to-Station protocol [Menezes, van Oorschot & Vanstone 1996] (using EC cryptographic primitives). Note: this is a key exchange protocol involving mutual authentication of the two parties (and also key authentication). In addition, each party's certificate is included in the exchange as no prior association of the devices is assumed.
   (a) A $\rightarrow$ B: $r_A P$, where r_A is a (temporary, secret) random value generated by A and P is the agreed fixed point on the (publicly known) elliptic curve.
   (b) B $\rightarrow$ A: $r_B P \mid\mid E_K(S_B(r_B P \mid\mid r_A P)) \mid\mid C_B$, where r_B is a (temporary, secret) random value generated by B, E_K denotes encryption using the key K derived from the shared secret $r_A r_B P$, S_X denotes a signature by party X using its device private key and C_X is the device certificate of X.
   (c) A: Decrypt and verify B's signature against it's provided certificate, and its certificate against the stored DIA public key.
   (d) A $\rightarrow$ B: $E_K(S_A(r_A P \mid\mid r_B P)) \mid\mid C_A$.
   (e) B: Decrypt and verify A's signature against its provided certificate, and its certificate against the stored DIA public key.
2. Derive session key from shared secret, and hence establish a mutually authenticated secure channel.

[9]RFC4122, http://www.ietf.org/rfc/rfc4122.txt

This protocol is implemented at an abstract level and can therefore be applied to any connection medium supported by Android. Notice that the protocol ensures that the connected device has been issued a certificate from the DIA and that they have access to the private key associated with that certificate. Furthermore the freshness of the exchange is guaranteed by the nonce $r_B P || r_A P$.

4.4 Security Discussion

In addition to the various security issues of each of the technologies used in connection initialisation, there are a number of additional issues that are described below.

4.4.1 Eavesdropping

Eavesdropping can occur during the NFC phase between the two devices. For example, Alice is using NFC to bootstrap the Bluetooth communication with Bob, and Eve eavesdrops on the MAC address and UUID passed over NFC. Once Eve has the MAC and UUID she can attempt to make a connection with Alice in place of Bob. If Eve is in possession of a legitimate device (or has access to signed certificates/private keys) she can fool Alice into communicating with her instead of Bob.

As discussed in Section 4.1 the NFC communication can be secured using a key exchange protocol. Alternatively, if future versions of Android support message response capabilities over NFC, the full station-to-station protocol could be completed via NFC. This would ensure that the device you tap against is the device you created a session key with.

4.4.2 Relay Attacks

Relay attacks, whereby a device acts as a proxy for a connection between a legitimate local target device and a legitimate remote target device (possibly via a secondary proxy in the remote location) would convince the local target device that it is interacting with it rather than the remote target device. Such attacks may be mitigated by use of:

- Dedicated hardware schemes for limiting the radius within which proxy forwarding may occur (*i.e.* based on the finite speed of light).
- Additional verification mechanisms, for example:
  - The display of common (random) numbers/imagery on each device, verified by each participant (less easy to relay by proxy).
  - Biometric authentication of each user to the other's device which is associated to a certificate linking each user's biometric to his/her device user ID. Note that this only ensures that the expected users are physically present, it does not protect against device proxying.

Note that we leave the mitigation of relay attacks for future work.

4.4.3 Replay Attacks

Replay attacks are where an adversary, Eve, eavesdrops on a key exchange protocol between communicating parties, Alice and Bob, and then uses the recorded key exchange protocol to mimic the parties at a later time. This gives the eavesdropper the ability to fool Bob into thinking that she is Alice, or vice versa. Replay attacks are mitigated in the station-to-station protocol by having the devices sign and encrypt the temporary secrets for verification. This means that unless the device that is being tricked happens to generate the same temporary secret twice it will be able to detect that the other device does not have access to the device private key. It is unlikely to generate the same temporary secret twice (assuming random number generation is truly random) because the minimum key size for this application is 256 bits, which provides $> 10^{77}$ possible values.

4.4.4 Denial of Service Attacks

Denial of service attacks are a broad category of attacks that aim to make a particular service or resource unavailable. Pelechrinis, Iliofotou & Krishnamurthy [2011] describe a number of jamming models, ranging from a constant jammer that transmits data constantly to more intelligent models that can selectively insert or destroy certain packets to disrupt communications. These kinds of attacks are not actively mitigated in this system, although given the use of an encrypted channel, utilising an intelligent jammer would be more difficult.

5 Session Management

On Android the life-cycle of an Activity[10] can be quite volatile and can change by way of user interaction, for example checking another application quickly, such as Short Message Service (SMS) or e-mail. This makes creating and storing the connection state inside an Activity difficult for the user to understand, especially when there are multiple applications that are part of the same application suite. Furthermore establishing a session is a relatively costly in terms of resources spent for both the devices and the user, therefore it is not ideal to create sessions constantly.

5.1 Implementation

The life-cycle problem may be solved by a session manager. The session manager is implemented as an Android service that is used to maintain connections across applications. This is done for a number of reasons, including:

- Usability – Users create connections once, which can be re-used for multiple tasks and are not closed when an Activity closes.

[10] An Activity is a basic application component for the Android platform; for more information see: http://developer.android.com/guide/components/activities.html

- Always listening – The service can listen for messages to a specific application and create notifications for the user. For example, if the session manager service receives a file send request it can show a notification alerting the user that someone wants to send a file to their device.
- Reduce permissions – Each application only has the minimum set of permissions required. A single permission to access SessionService means that applications can make use of Bluetooth/NFC without directly having access to those features.
- Defined Interface – SessionService mediates other application's access to its features through a well-defined interface that includes methods for tasks such as sending messages, and registering receivers.
- Re-use – The session service can be reused by applications without having to alter any of the existing code. This means new applications that rely on ad hoc networking can be quickly developed rather than having to develop the ad hoc networking code themselves. Note that this depends on the Android API versions being constant among the applications.
- Modular design – The session service supports the idea of modules that can be hooked into the existing lifecycle of the service. This allows modules to act when events occur, for example to create a notification when a message is received.

Unfortunately there are some drawbacks that affect the usefulness of the session service. These issues will influence the potential use-cases that can be implemented using the session service.

- Inter-Process Communication (IPC) – Restricts the size of messages sent over the session service because IPC must be used for every message. Currently this buffer is set to 1Mb which is shared by all transactions that are currently in progress for that specific process [*TransactionTooLargeException* 2013].
- Higher resource utilisation – Running the service in another process consumes more random-access memory (RAM) and central processing unit (CPU) time. This could be a problem for low specification devices with low RAM or a slow CPU.
- User Error – Users may leave the sessions open after they are finished. This increases the time-frame within which a device can be subject to attack, since it is still listening and communicating over Bluetooth/Wi-Fi. This means that any vulnerabilities in the communication stacks have more time to be exploited by an adversary.
- Speed – Because of the level of abstraction and IPC performed, the session service is somewhat slower than performing this work in the same process.

5.1.1 Activity

The session manager has a set of Activities that are responsible for handling the initialisation of the connections. This set includes an NFC Activity that is used to transfer the required details for the session service to begin the connection protocol. This Activity is registered to not only use Android Beam to push data, but also to listen for NFC pushes. This means that the same Activity can do either, depending on whichever person activates the NFC beam first. The steps for transmitting the connection details via NFC are as follows:

1. A: Retrieve Bluetooth MAC address from Bluetooth adapter and generate new random UUID.
2. A $\rightarrow$ B: Push MAC address and random UUID pair.
3. A: Listen for connection using the UUID.
4. B: Make active connection to MAC address on the service UUID provided.

Note that listening and active connection creation occur within the session service, and is described in Section 5.1.2.

5.1.2 Service

The service can create Bluetooth connections or listen for connections. The way it acts depends on the method called by the initiator Activity. Once a connection has been initialised the session service queries the other device for an identity to associate to the session. An identity is simply a wrapper for a device certificate that provides relevant information about the owner of the device. This allows other applications to treat the session as an individual rather than a connection. It also provides another point of verification for the user, because identities can contain information such as a photo, name, and date of birth that the user can verify.

The session service was created with a focus on re-use and modularity, thus it supports a set of hooks that allow modules to be added to it dynamically in order to add behaviour when certain events occur. For example, there is a module which displays a notification to the user, alerting them to the number of devices that are currently connected to the service. Another module, for instance, could scan messages for the appropriate subject field and start an activity or create a notification based on that subject field. A module that was implemented as part of this project is outlined in Section 5.2.

5.2 Peer Discovery

One module we created for the session service was a peer discovery module. This module is responsible for dynamically discovering other peers in the network as they connect. It is useful because of the way Android's Bluetooth capabilities are used, rather than having one single device that hosts the network, we create a large set of individual connections between devices. Within these connections either device may be hosting that particular link. This is required for a number of reasons including:

- Minimise the attack surface by restricting the time spent listening for connection to those times we expect a connection.

- Increase usability by allowing anybody to initiate connections with anyone else in order to be added to the network. This is especially important if there are a large number of people linking their devices together.

Peer discovery begins when a new connection is detected. Both sides send an ‘initiate peer discovery’ message and reply to the message with a list of their currently connected peers (as represented by their identity). Once a list of peers is received, the receiver then searches through the list to find any peers that it is not currently connected to. When one is found the process performed is shown in Figure 4.

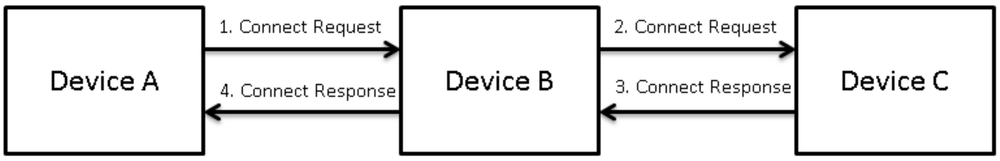

Figure 4: *Peer discovery process.*

Note that Device A and Device C are both already connected to, and aware of Device B. Once this process is complete Device A initiates a direct connection to Device C using the information provided in the connect response.

There are a few disadvantages to peer discovery:

- Adds small overheads when receiving messages because it needs to determine whether or not a message is a peer discovery message.
- Adds small overhead when performing initial connection because it needs to perform peer discovery in addition to the standard connection procedures.
- Moderately slow to find and connect to peers after initial connection (2-3 seconds).
- Relies on the Device B being a legitimate device and not altering the details provided by Device A or C.

5.3 Security Discussion

While the session service provides a wide range of benefits to the overall system in terms of usability and reuse, there are a number of potential security issues that arise as a result of the design of the session service. For example:

- Signature level permissions are shared by all applications signed by the same signer. This is not ideal because it could lead to potentially dangerous permission combinations (for instance the session manager could gain write access to the signing service, which means that the session manager could add/replace credentials).

- If the Android system service is compromised, unencrypted traffic will be readable by the adversary as IPC calls. This is because encryption occurs inside the session service.

Such issues can be mitigated through the use of a strong application boundaries, and permission schemes, as well as a good security review process. The SE Android kernel and middleware MAC can be used to enforce the application boundaries and permission schemes, as well as help prevent root exploits that could allow system applications to become compromised.

6 Future Work

There are a number of potential areas for improvement identified throughout this report, pertaining to both this specific project and also for some of the technologies utilised. Some of these possible avenues of improvement or further research are:

- Card Emulation to achieve full key exchange over NFC[11] (host-based card emulation was introduced in Android 4.4).
- Investigate potential USB-to-USB connectivity[12] for more sensitive transactions. Whether or not this is possible with the current platforms is unknown at this stage.
- Investigate the use of IR in place of NFC[13] to minimise eavesdropping. Though there are not many Android devices with IR support released at the time of writing, the introduction of the API indicates that hardware supporting it will be released in the future.
- Haselsteiner & Breitfuß [2006] describe an NFC specific key-exchange protocol that could be used to encrypt NFC communications efficiently. This would bolster the overall security of NFC and help to eliminate threats such as eavesdropping.
- Implementation of a biometric authentication method whereby user A authenticates himself/herself to user B's device to eliminate man-in-the-middle (MITM) attacks. For example, facial recognition, which could make use of the forward or rear facing cameras available on most Android devices, may be used.
- Investigate the potential for utilising the secure element in Android devices to store sensitive information (for instance, private keys). This would involve working with the handset manufacturers in order to gain access to the secure element.

[11] http://developer.android.com/guide/topics/connectivity/nfc/hce.html
[12] http://developer.android.com/tools/adk/index.html
[13] http://developer.android.com/about/versions/android-4.4.html

References

Agrawal, P. & Bhuraria, S. (2012) Near field communication, *IT Matters* **67**.

Android Debug Bridge (2013) URL – `http://developer.android.com/tools/help/adb.html`. [Accessed: 18-December-2013].

Android Security Overview (2013) URL – `http://source.android.com/tech/security/`. [Accessed: 18-December-2013].

Beaumont, M., Hopkins, B. & Newby, T. (2011) *Hardware Trojans-Prevention, Detection, Countermeasures (A Literature Review)*, Technical report, DTIC Document.

Elenkov, N. (2013) Credential storage enchancements in Android 4.3. URL – `nelenkov.blogspot.co.uk/2013/08/credential-storage-enhancements-android-43.html`.

Felt, A. P., Wang, H. J., Moshchuk, A., Hanna, S. & Chin, E. (2011) Permission re-delegation: Attacks and defenses., *in USENIX Security Symposium*.

Fledel, Y., Shabtai, A., Potashnik, D. & Elovici, Y. (2012) Google Android: An Updated Security Review, *Mobile Computing, Applications, and Services* pp. 401–414.

Fora, P. O. (2013) Defeating security enhancements (se) for android, *in DefCon21*. URL – `http://www.defcon.org/images/defcon-21/dc-21-presentations/Fora/DEFCON-21-Fora-Defeating-SEAndroid.pdf`.

Franceschi-Bicchierai, L. (2012) Darpa starts up-armoring android phones, tablets. URL – `http://www.wired.com/dangerroom/2012/06/armored-android/`.

Friedman, R., Kogan, A. & Krivolapov, Y. (2013) On power and throughput tradeoffs of wifi and bluetooth in smartphones, *Mobile Computing, IEEE Transactions on* **12**(7), 1363–1376.

Getting Started (2013) URL – `http://developer.android.com/training/index.html`. [Accessed: 18-December-2013].

Gleeson, A. & Lucas, M. (2013) *Security of the Android Operating System*, General Document DSTO-GD-0779, DSTO.

Haselsteiner, E. & Breitfuß, K. (2006) Security in near field communication (nfc), *in Workshop on RFID Security RFIDSec*.

Khan, S., Banuri, S. H. K., Nauman, M., Khan, S. & Alam, M. (2010) Analysis report on Android Application Framework and existing Security Architecture.

Kizza, J. M. (2013) Security in wireless networks, *in Guide to Computer Network Security*, Springer, pp. 387–411.

Kortvedt, H. S. & Mjølsnes, S. F. (2009) Eavesdropping near field communication, *in The Norwegian Information Security Conference (NISK)*.

Liebergeld, S. & Lange, M. (2013) Android security, pitfalls and lessons learned, *in Information Sciences and Systems 2013*, Springer, pp. 409–417.

Loscocco, P. & Smalley, S. (2001) Integrating flexible support for security policies into the linux operating system., *in USENIX Annual Technical Conference, FREENIX Track*, USENIX, pp. 29–42. URL – `http://dblp.uni-trier.de/db/conf/usenix/usenix2001f.html#LoscoccoS01`.

Lu, Y., Meier, W. & Vaudenay, S. (2005) The conditional correlation attack: A practical attack on bluetooth encryption, *in Advances in Cryptology–CRYPTO 2005*, Springer, pp. 97–117.

Menezes, A., van Oorschot, P. C. & Vanstone, S. A. (1996) *Handbook of Applied Cryptography*, CRC Press.

Padgette, J., Scarfone, K. & Chen, L. (2012) Guide to bluetooth security, *NIST Special Publication* **800**, 121.

Pelechrinis, K., Iliofotou, M. & Krishnamurthy, S. V. (2011) Denial of service attacks in wireless networks: The case of jammers, *Communications Surveys & Tutorials, IEEE* **13**(2), 245–257.

Permission Element (2013) URL – `http://developer.android.com/guide/topics/manifest/permission-element.html`. [Accessed: 18-December-2013].

Remedios, D., Sousa, L., Barata, M. & Osório, L. (2006) Nfc technologies in mobile phones and emerging applications, *in Information Technology For Balanced Manufacturing Systems*, Springer, pp. 425–434.

Rosenberg, D. (2013) Unlocking the Motorola bootloader. URL – `http://blog.azimuthsecurity.com/2013/04/unlocking-motorola-bootloader.html`.

SE Android Notebook (2013) URL – `http://selinuxproject.org/page/NB_SEforAndroid_2`.

Smalley, S. (2011) The case for se android. URL – `http://selinuxproject.org/~jmorris/lss2011_slides/caseforseandroid.pdf`.

Smalley, S. (2012) Security Enhanced (SE) Android, *LinuxCon North America, San Diego, USA* .

Smalley, S. (2013) Laying a Secure Foundation for Mobile Devices, *NDSS* .

Smalley, S. & Craig, R. (2013) Security enhanced (se) android: Bringing flexible mac to android, *in Network & Distributed System Security Symposium (NDSS13)*.

Spill, D. & Bittau, A. (2007) Bluesniff: Eve meets alice and bluetooth, *in Proceedings of USENIX Workshop on Offensive Technologies (WOOT)*.

The AndroidManifest.xml File (2013) URL – `http://developer.android.com/guide/topics/manifest/manifest-intro.html`. [Accessed: 18-December-2013].

TransactionTooLargeException (2013) URL – `http://developer.android.com/reference/android/os/TransactionTooLargeException.html`. [Accessed: 18-December-2013].

Validating Security-Enhanced Linux in Android (2013) URL – `http://source.android.com/devices/tech/security/se-linux.html`. [Accessed: 18-December-2013].

Van Damme, G., Wouters, K. & Preneel, B. (2009) Practical experiences with nfc security on mobile phones, *in Workshop on RFID Security–RFIDSec09*.

Appendix A Certificate Authority Key Generation

```
C:\>keytool -genkey -alias dia.ca.example -keyalg RSA -keysize 2048
    -keystore H:\CAKeyStore.jks
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  Device Issuing Authority
What is the name of your organizational unit?
  [Unknown]:  Cyber and Electronic Warfare Division
What is the name of your organization?
  [Unknown]:  Defence Science and Technology Organisation
What is the name of your City or Locality?
  [Unknown]:  Adelaide
What is the name of your State or Province?
  [Unknown]:  South Australia
What is the two-letter country code for this unit?
  [Unknown]:  AU
Is CN=Device Issuing Authority, OU=Cyber and Electronic Warfare Division,
    O=Defence Science and Technology Organisation, L=Adelaide, ST=South
    Australia, C=AU correct?
  [no]:  yes

Enter key password for <dia.ca.example>
        (RETURN if same as keystore password):

C:\>
```

Appendix B CSR to CA Signed Certificate

```
protected final X509Certificate generateSignedCertificate(
        X500Name pSubjectName,
        SubjectPublicKeyInfo pSubjectPublicKeyInformation) {
    // Calculate the start/end dates for this certificate
    Calendar lCalander = Calendar.getInstance();
    Date lStartDate = lCalander.getTime();
    // Certificate lasts for 10 years
    lCalander.set(Calendar.YEAR, lCalander.get(Calendar.YEAR) + 10);
    Date lExpiryDate = lCalander.getTime();

    // Serial is just an ever increasing number
    BigInteger lSerialNumber = new BigInteger(mCurrentSerial);

    // Generate the Certificate
    // - Issuer is me (CA)
    // - Subject is the requester
    X509v3CertificateBuilder lGen = new X509v3CertificateBuilder(
            getCAName(), lSerialNumber, lStartDate, lExpiryDate,
            pSubjectName, pSubjectPublicKeyInformation);

    // Create the content signer and sign the Certificate
    JcaContentSignerBuilder lContentSignerBuilder =
            new JcaContentSignerBuilder("SHA512WithRSA");

    // Create the ContentSigner with the CA's private key
    ContentSigner lContentSigner = lContentSignerBuilder
            .build(getCAPrivateKey());

    // Sign the certificate
    X509CertificateHolder lTempResult = lGen.build(lContentSigner);

    return extractCertificate(lTempResult);
}

protected final X509Certificate extractCertificate(
        X509CertificateHolder pHolder) {
    X509Certificate lResult = null;
    InputStream lCertificateStream;

    CertificateFactory lCertFactory = CertificateFactory
            .getInstance("X.509");
    // Read user Certificate
    lCertificateStream = new ByteArrayInputStream(pHolder.getEncoded());
    lResult = (X509Certificate) lCertFactory
            .generateCertificate(lCertificateStream);
        lCertificateStream.close();
        return lResult;
    }
```

Appendix C SE Android Install-time MAC Policy

Constructed using reference policy from *SE Android Notebook* [2013].

```
<signer signature="@DSTO">
    <seinfo value="dsto"/>
    <package name="dsto.connectivity.sessionmanager">
        <seinfo value="dsto"/>
        <allow-permission name="android.permission.BLUETOOTH"/>
        <allow-permission name="android.permission.BLUETOOTH_ADMIN"/>
        <allow-permission name="android.permission.NFC"/>
        <allow-permission name="dsto.connectivity.verifyingservice
            .permission.READ_PERMISSION"/>
        <allow-permission name="dsto.connectivity.signingservice
            .permission.READ_PERMISSION"/>
    </package>

    <package name="dsto.connectivity.deviceprovisioner">
        <seinfo value="dsto"/>
        <allow-permission name="dsto.connectivity.verifyingservice
            .permission.READ_PERMISSION"/>
        <allow-permission name="dsto.connectivity.verifyingservice
            .permission.WRITE_PERMISSION"/>

        <allow-permission name="dsto.connectivity.signingservice
            .permission.READ_PERMISSION"/>
        <allow-permission name="dsto.connectivity.signingservice
            .permission.WRITE_PERMISSION"/>
    </package>
</signer>

<default>
    <seinfo value="default"/>
    <deny-permission name="dsto.connectivity.verifyingservice
        .permission.WRITE_PERMISSION"/>
    <deny-permission name="dsto.connectivity.verifyingservice
        .permission.READ_PERMISSION"/>

    <deny-permission name="dsto.connectivity.signingservice
        .permission.WRITE_PERMISSION"/>
    <deny-permission name="dsto.connectivity.signingservice
        .permission.READ_PERMISSION"/>
</default>
```

THIS PAGE IS INTENTIONALLY BLANK

Page classification: UNCLASSIFIED

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION DOCUMENT CONTROL DATA					1. CAVEAT/PRIVACY MARKING	
2. TITLE Secure Ad Hoc Networking on an Android Platform			3. SECURITY CLASSIFICATION Document (U) Title (U) Abstract (U)			
4. AUTHORS Angus Morton, David Adie, Paul Montague			5. CORPORATE AUTHOR Defence Science and Technology Organisation PO Box 1500 Edinburgh, South Australia 5111, Australia			
6a. DSTO NUMBER DSTO–TN–1390	6b. AR NUMBER 016-201		6c. TYPE OF REPORT Technical Note		7. DOCUMENT DATE May, 2014	
8. FILE NUMBER 2013/1248018/1	9. TASK NUMBER INT 07/012	10. TASK SPONSOR CIOG		11. No. OF PAGES 25		12. No. OF REFS 35
13. URL OF ELECTRONIC VERSION `http://www.dsto.defence.gov.au/publications/scientific.php`			14. RELEASE AUTHORITY Chief, Cyber and Electronic Warfare Division			
15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT *Approved for Public Release* OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE, PO BOX 1500, EDINBURGH, SOUTH AUSTRALIA 5111						
16. DELIBERATE ANNOUNCEMENT No Limitations						
17. CITATION IN OTHER DOCUMENTS No Limitations						
18. DSTO RESEARCH LIBRARY THESAURUS Mobile computing, Secure communications, Information security						
19. ABSTRACT This document describes and analyses a suite of applications for the Android platform that enables easy and secure ad hoc networking, with a focus on extensibility and reusability. The applications were developed by the first author during a Swinburne University Industry-Based Learning placement at the Defence Science and Technology Organisation, under the latter's Industry Experience Placement Program.						

Page classification: UNCLASSIFIED